Francis Jingo, Sonali Pandit, Lanston Chen, Jean Baptiste Habyarimana, Alejandro Chumaceiro

ISOM-671: Managing Big Data

November 17th, 2023

Group Assignment 2: Hadoop, Hive, and Pig

Q1. Hive v. Pig Data Migration Cost

This report presents an analysis for migrating 500TB of data to an HDFS platform, considering datanode requirements and associated costs. The analysis assumes each datanode has a storage capacity of 64TB, with 25% reserved for intermediate tasks, and data growth at 4% per quarter. AWS's D3en 4xlarge EMR instance, costing $157.68 per node per month, is used for cost estimation.

Datanode Requirement Calculation: The effective storage capacity per datanode, considering 25% usage for intermediate tasks, is calculated as: Effective Storage = Total Storage × (1 − Intermediate Task Ratio). For a 64TB node: Effective Storage = 64T B × 0.75 = 48T B

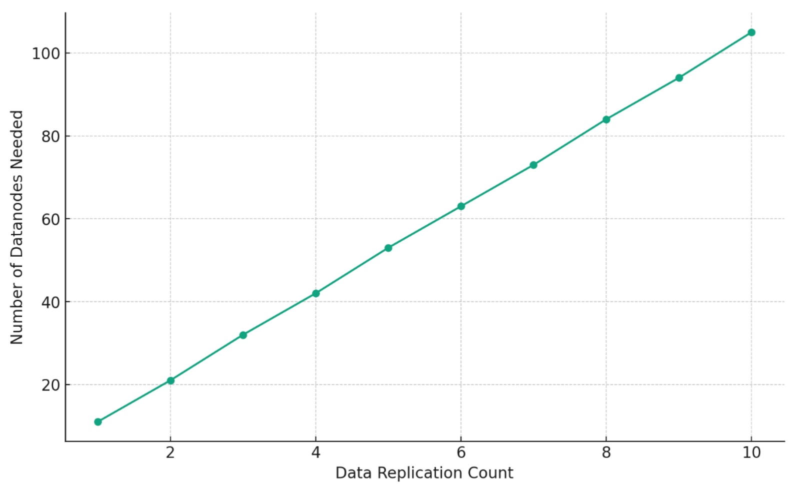


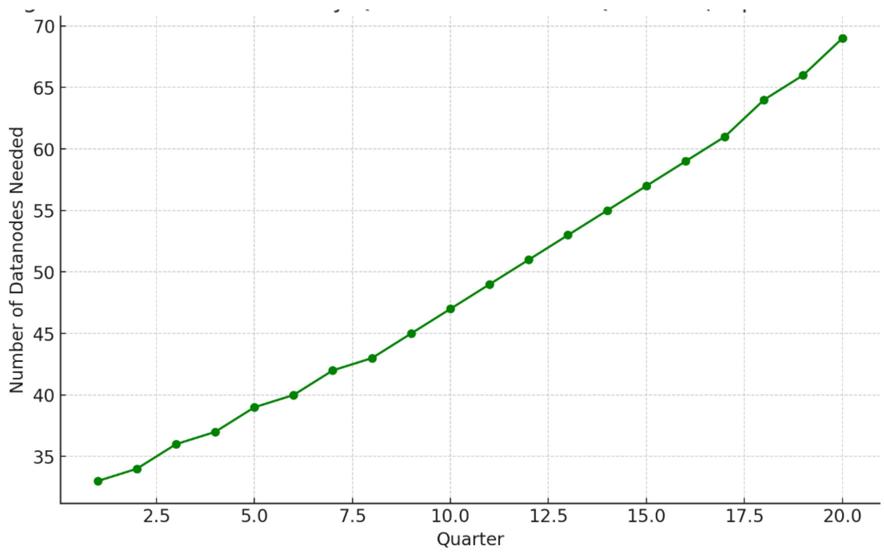
Figure 1: Datanodes Needed based on Data replication count plot

Datanode Calculation for Replication Factor 3 :With a replication factor of 3 and data growth of 4% per quarter, the number of datanodes required over 20 quarters is calculated as:

$$\text{Datanodes Required} = \lceil \frac{\text{Data Size} \times \text{Replication Factor}}{\text{Effective Storage}} \rceil$$

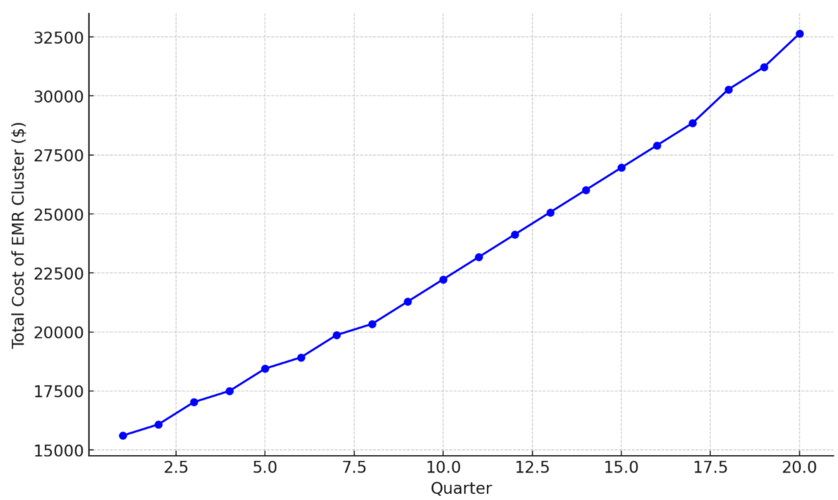
Data size increases each quarter as:

$$\text{Data Size}_{\text{new}} = \text{Data Size}_{\text{previous}} \times (1 + \text{Growth Rate})$$



EMR Cluster Cost Over Next 20 Quarters

Cost Estimation: The cost per quarter for each datanode is $473.04 (3 months at $157.68 per month). The total cost for the cluster is calculated as: Total Cost = Cost per Datanode per Quarter × Number of Datanodes.An assumption of a linear increase in price is made for the cost estimation. This accounts for potential increases in operational costs, inflation, or changes in AWS pricing over time. The total cost over 20 quarters is estimated using the number of datanodes required and the cost per datanode, considering the data growth rate and the assumption of linear price increase.



EMR Cluster Cost Over Next 20 Quarters

The analysis provides an infrastructure and cost framework for data migration to HDFS. It emphasizes the need for considering data growth, replication, and storage utilization in migration planning. The use of AWS's D3en 4xlarge offers a conservative cost estimate, aiding in budgeting and resource planning.

Q2. Data Migration into Hive

Differences

Approach	Difference
Moving data from MySQL to Hive directly using Sqoop	Has the fastest execution time because it does not have intermediate storage steps. Data moves directly to Hive using Sqoop which optimizes the movement.
Moving data from MySQL to S3 using Sqoop, then using LOAD DATA HQL to move it to Hive	Involves moving data from Mysql to S3 and then loading it into Hive. It is slower than the first approach. Sqoop optimizes the movement therefore in cases of big data files, loading the data using hql might be slower.
Loading a CSV to S3 and using *LOAD DATA HQL* to move it to Hive	Involves loading csv data to S3 and then loading it into Hive. This approach was relatively fast. However for cases when there is a lot of data csv file like a million records, loading the CSV may take extensively long making it the slowest approach.

Use Case

Approach	Use case
Moving data from MySQL to Hive directly using Sqoop	When performance is critical and we need to minimize data transfer time. Also when there is no need for intermediate storage or staging.
Moving data from MySQL to S3 using Sqoop, then using LOAD DATA HQL to move it to Hive	When we need to perform intermediate processing on the data before loading it into Hive. and when we need to decouple storage steps from the processing steps.
Loading a CSV to S3 and using *LOAD DATA HQL* to move it to Hive	When working with data for archival purposes and having to load data into Hive without using Sqoop.

Q3. Hive v. Pig

Use Case: Based on the findings in the paper, Hive outperforms Pig when aggregation is a requirement for analysis, especially between larger sized files. Furthermore, the paper argues that Hive should be used for business intelligence processes and analysis while Pig should be used for ETL. To illustrate, to perform the same join and aggregation query, Pig requires extensive operations whereas Hive requires minimal statements.

Variation in Execution Times: In this assignment we attempted to replicate the experiment within Kendal et al.'s paper by running an identical query 10 times within Hive and Pig for the same sized dataset. From our findings, on average, HiveQL outperformed Pig by approximately 3 seconds within the 10 iterations. While Pig had the fastest execution time, Hive consistently outperforms Pig due to having a lower average execution time and significantly lower variation, as seen from Hive's standard deviation within the sample of 10 queries. Pig is designed to perform more efficiently as the degree of decentralization increases, meaning that data is highly distributed across nodes.

Execution Times (sec)

Query #	Pig	Hive
1	22	19.957
2	21	15.842
3	14	16.923
4	20	15.807
5	17	15.095
6	15	14.474
7	16	14.395
8	23	15.679
9	15	14.264
10	23	14.889
Mean	18.6	15.7325
Max	23	19.957
Min	14	14.264
St. Dev	3.382307	1.610317
Count	80359	82754

Differences to Paper: Like the research paper Hive outperformed Pig in terms of execution times and length of query in order to achieve the same output. Alternatively, in our replication of the experiment, Pig had the lowest execution time of approximately 14 seconds whereas in the research paper Pig did not approach Hive's faster execution time. The execution output from Pig did not demonstrate time in the microsecond scale, hence the reason why Pig's execution times only demonstrate time at the seconds level. Moreover, the difference in magnitude between Hive and Pig's standard deviation was much higher in our replication when compared to the paper. However, this difference may be attributable to sample size of iterations.

Appendix

Question 2 Code

```
-- Create three happiness tables (happy1, happy2, and happy3) in Hive
-- on Hive
create database happiness;
-- based on 2016
create table happiness.happy1 (Country varchar(30), Region varchar(50), `Happiness_Rank`
int, `Happiness Score` double, `Lower Confidence Interval` double, `Upper Confidence Interval`
double, `Economy (GDP per Capita)` double, Family double, `Health (Life Expectancy)` double,
Freedom double, `Trust (Government Corruption)` double, Generosity double, `Dystopia
Residual` double) row format delimited fields terminated by ',' lines terminated by '\n';
create table happiness.happy2 (Country varchar(30), Region varchar(50), `Happiness_Rank`
int, `Happiness Score` double, `Lower Confidence Interval` double, `Upper Confidence Interval`
double, `Economy (GDP per Capita)` double, Family double, `Health (Life Expectancy)` double,
Freedom double, `Trust (Government Corruption)` double, Generosity double, `Dystopia
Residual` double) row format delimited fields terminated by ',' lines terminated by '\n';
create table happiness.happy3 (Country varchar(30), Region varchar(50), `Happiness_Rank`
int, `Happiness Score` double, `Lower Confidence Interval` double, `Upper Confidence Interval`
double, `Economy (GDP per Capita)` double, Family double, `Health (Life Expectancy)` double,
Freedom double, `Trust (Government Corruption)` double, Generosity double, `Dystopia
Residual` double) row format delimited fields terminated by ',' lines terminated by '\n';

-- Load csv file in RDS mySQL (data1)
create schema happiness;
create table happiness.data1(
Country varchar(30), Region varchar(50), `Happiness_Rank` int, `Happiness Score` double,
`Lower Confidence Interval` double, `Upper Confidence Interval` double, `Economy (GDP per
Capita)` double, Family double, `Health (Life Expectancy)` double, Freedom double, `Trust
(Government Corruption)` double, Generosity double, `Dystopia Residual` double);

LOAD DATA LOCAL
  INFILE '/Users/FrancisJingo1/Downloads/worldhappiness/2016.csv'
  INTO TABLE happiness.data1
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\b'
```

```
  LINES TERMINATED BY '\n' IGNORE 1 ROWS;

-- Using Sqoop import happiness data (data1) from RDS to S3
sqoop import --connect
jdbc:mysql://mysql-rds-lab5.c8sutaslqpmn.us-east-1.rds.amazonaws.com/happiness
--username admin --password admin1234 --delete-target-dir --target-dir
s3://group-assignment-2/data1/ --query 'select * from data1 where $CONDITIONS' --split-by
data1.Happiness_Rank;
-- On Hive
-- and then use HiveQL to import that S3 data in Hive (happy1)
LOAD DATA INPATH 's3://group-assignment-2/data1/' INTO TABLE happiness.happy1;
-- Using Sqoop, import happiness data (data1) from RDS to Hive (happy2)
sqoop import --connect
jdbc:mysql://mysql-rds-lab5.c8sutaslqpmn.us-east-1.rds.amazonaws.com/happiness
--username admin --password admin1234 --query 'select * from data1 where $CONDITIONS'
--split-by data1.Happiness_Rank --fields-terminated-by ',' --hive-import --hive-database
happiness --hive-table happy2 --delete-target-dir --target-dir
/user/hive/warehouse/happiness.db/happy2;
-- Using HiveQL, import S3 data (data2) in Hive (happy3).
LOAD DATA INPATH 's3://group-assignment-2/data2/2016.csv' INTO TABLE happiness.happy3;
ALTER TABLE happiness.happy3 SET TBLPROPERTIES ("skip.header.line.count"="1");
```

When dealing with the importation of data into Hive, there are various approaches one can take, each suited to different use cases and requirements. Here, we'll discuss three specific scenarios:

Using Sqoop to Import Data from RDS to S3, Then Importing to Hive with HiveQL (Data1 to Happy1):

- Use Case: This approach is ideal when dealing with very large datasets that need intermediate storage or when there's a need to keep a raw data backup in S3. S3 provides a cost-effective, scalable storage solution. By first moving data to S3, you ensure that the data is available in a raw format for other uses beyond Hive. This method is also useful if the data in S3 is to be used by other systems or for analytics beyond Hive.

- Workflow: Data is first imported from an RDS database to an S3 bucket using Sqoop. Once in S3, the data is then imported into Hive using HiveQL. This two-step process allows for more flexibility in data handling and storage.

Using Sqoop to Directly Import Data from RDS to Hive (Data1 to Happy2):

- Use Case: This method is straightforward and efficient for scenarios where the primary objective is to perform analytics using Hive. It's suitable when there's no requirement to store the raw data in a scalable storage like S3 or when the data volume is not excessively large. This approach simplifies the workflow by eliminating the intermediate step of storing data in S3.
- Workflow: Data is imported directly from RDS into Hive using Sqoop. This approach is more streamlined and is optimal for quicker data availability in Hive for analysis, as it reduces the time and complexity involved in transferring data.

Using HiveQL to Import Data from S3 into Hive (Data2 to Happy3):

- Use Case: This approach is ideal when the data is already stored in S3, possibly as a result of previous data pipelines, or when using S3 as a data lake. It's suitable in environments where S3 serves as the central repository for various data sources and Hive is used specifically for data querying and analysis.
- Workflow: Here, HiveQL is used to directly import data from S3 into Hive. This method bypasses the need for Sqoop and is preferred when dealing with data already residing in S3. It's an efficient way to integrate Hive with existing data stored in S3, leveraging the storage and scalability benefits of S3.

Each of these approaches has its merits and is best suited to specific scenarios depending on factors like data size, the complexity of the data pipeline, storage preferences, and the end goal of the data analysis.

Question 3 Code

HQL

```
-- Creating database to store data
CREATE DATABASE retail;

-- Column definitions (refer to data for column names and data types)
CREATE TABLE online_retail
(InvoiceNo INT,
StockCode STRING,
Description STRING,
Quantity INT,
InvoiceDate TIMESTAMP,
UnitPrice DOUBLE,
CustomerID INT,
Country STRING
```

```
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n';

-- Load data
LOAD DATA INPATH 's3://datasets-achumac/Online Retail/online-retail-dataset.csv' OVERWRITE INTO TABLE online_retail;

-- Query: execute a query to identify the number of orders with UnitPrice>5 and Quantity<10
SELECT count(*) FROM online_retail where UnitPrice>5 and Quantity<10;
```

Pig

```
-- Enter Pig (grunt)
pig

-- Load the data
retail = LOAD 's3://datasets-achumac/Online Retail/online-retail-dataset.csv' USING PigStorage(',') AS (InvoiceNo: int, StockCode: chararray, Description: chararray, Quantity: int, InvoiceDate: chararray, UnitPrice: double, CustomerID: int, Country: chararray);


-- Filter the sales
filtered = FILTER retail BY UnitPrice>5 AND Quantity<10;

-- Group
retail_groupped = GROUP filtered ALL;

-- Count
count = FOREACH retail_groupped GENERATE COUNT(filtered.InvoiceNo);

-- dump count;
```